Software build (-erstellung), deployment(-verteilung) und execution(-ausführung)
(in eingebetteten Systemen)

Inputvortrag „Nebenläufige Systeme“ am 1.11.18

Georg Ringwelski

Was machen Sie beim Entwickeln von Software?

- Programmieren → Handarbeit
- Übersetzen → build
- GGf. auf Zielplattform kopieren → deploy
- Ausführen → execute

Was ist der „Build-Prozess“

- Teilaufgabe der Softwareentwicklung: Erstellung des ausführbaren Systems, von Tests, Dokumentation etc
- Automatisiert und dadurch leicht wiederholbar
- Bei Anwendungserstellung meistens zwei Aufgaben:
  - Kompilieren

    zB javac –cp=… -d=… de\hszg\myPackage*.java

    → vgl. Optionen für Java-Compiler
  - Linken an Bibliotheken

    zB nxjlink –o nxt\MyProg.nxj de.hszg.MyProg

Zettelwand

Welche Informationen sind für einen erfolgreichen build-Prozess notwendig?

(und gehören damit in die formale Beschreibung des Prozesses)

Build-Prozess:

Ein automatisierter Prozess zur Steuerung von Befehlen nach Abhängigkeiten um Ziele zu erreichen

- Beispiele für Befehle:
  - nxjc Hello.java → Java class erzeugen
  - nxjlink –o Hello.nxj Hello → linken für NXT
  - nxjupload –r Hello.nxj → auf NXT installieren und starten
- Beispiele für Abhängigkeiten
  - Hello.java → Die Datei muss vorhanden sein/geändert worden sein
- Beispiele für Ziele:
  - Link → Kompilieren und übersetzen
  - Install → auf NXT hochladen
  - runPC →auf Computer ausfüren
  - Test → UnitTests ausführen

Beobachtung:
Auch bei deploy und execute geht es um Ziele, Abhängigkeiten und Befehle

Übungsaufgabe 3

(**build-**) Ziele, Abhängigkeiten und Befehle?

1. Ziele: GUI-Anwendung erstellen, Test1 erstellen (.class Dateien)
2. Abhängigkeiten: Ihr Java-Code, zB jfxrt.jar, junit.jar, …
3. Befehle: javac …

(**deployment-**) Ziele: .class Dateien im richtigen Verzeichnis

(**execute-**) Ziele, Abhängigkeiten, Befehle:

1. Ziele: Test1 ausführen, GUI starten
2. Abhängigkeiten: .class/.jar-Dateien im richtigen Verzeichnis
3. Befehle: java …

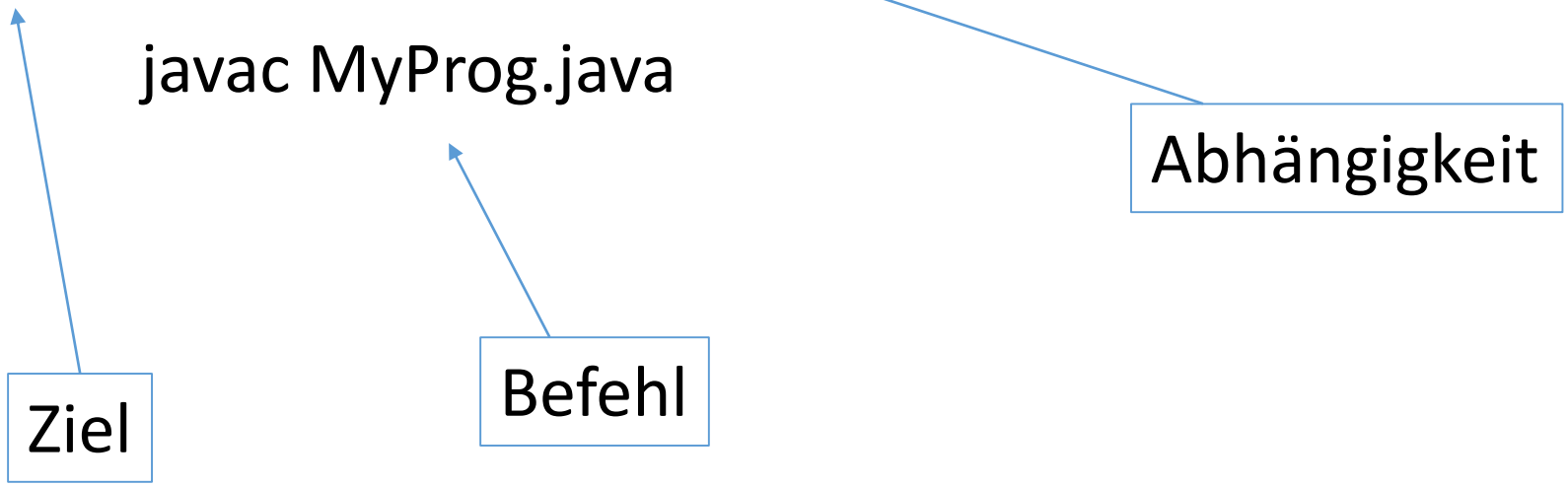
HOWOTO build, deploy, execute

Programm übersetzen	Programm installieren	Programm ausführen
Build configuration in IDE	Deployment mit IDE (manchmal unterstützt, manchmal nicht)	Run configuration in IDE
Tools: make, ant, maven, buildr, gradle etc	Tools: jenkins, Zenworks (Java WebStart)	Auch mit build tools
Kommandozeile, shellscript oder batch-datei	Kopieren, Installationprogramme, Skripte, OS-spezifisch	Kommandozeile, per Mausklick im OS, shellscript

Nightly Build → Continuous Integration → Continuous Delivery

Beispiel: make (Datei: Makefile)

```
runNXT: MyProg.nxj
        nxjupload –r MyProg.nxj
MyProg.nxj: MyProg.class
        nxjlink –o MyProg.nxj MyProg
MyProg.class: MyProg.java
        javac MyProg.java
```



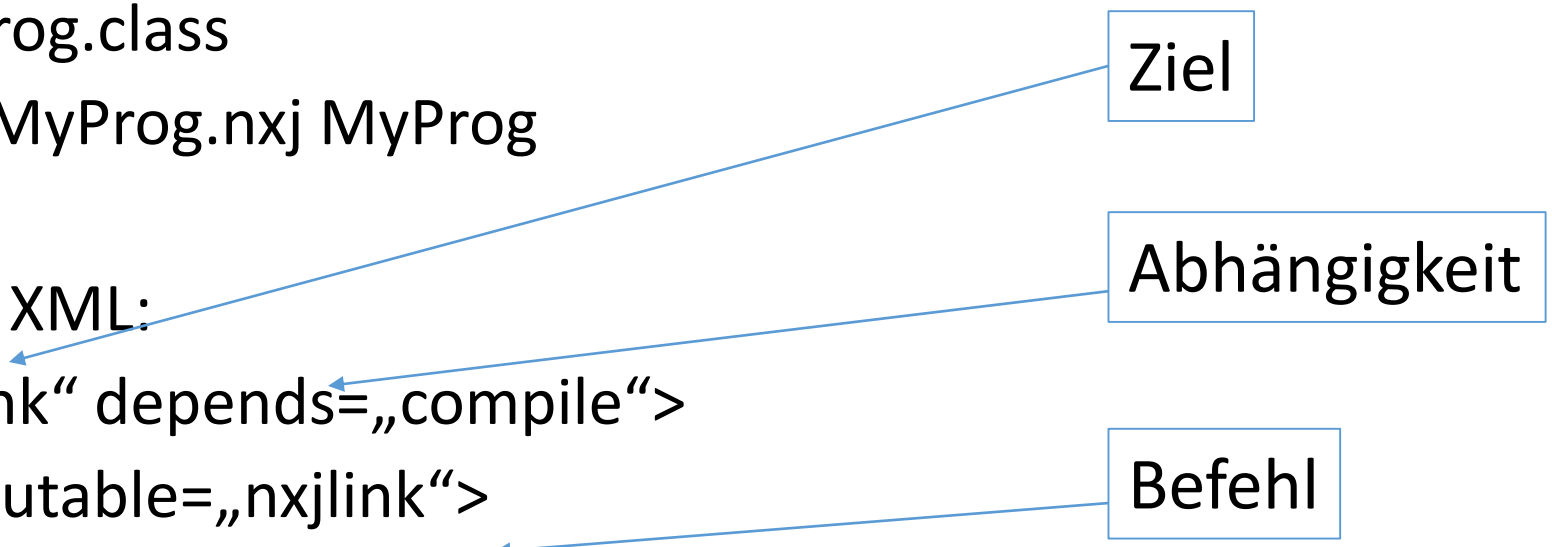
Von make zu ant (Datei: build.xml)

z.B.

```
MyProg.nxj: MyProg.class
        nxjlink –o MyProg.nxj MyProg
```

Wird übersetzt in XML:

```
<target name=„link“ depends=„compile“>
        <exec executable=„nxjlink“>
                <arg line=„-o MyProg.nxj MyProg/>
        </exec>
</target>
```



Build tools im Wandel der Zeiten

1977: make, in Unix/Linux immer noch Standard

2000: ant, sehr schnell sehr erfolgreich in der Java Welt

- Vorteile: einfach, Nachteil: Hierarchische und umfangreiche Struktur von XML
- 2006: Erweiterung für verteilte Entwicklung: Ivy

2004: maven: „Konvention über Konfiguration“ in Java

- : Es bleibt nur die Festlegung der Ziele (XML), außer man möchte mal was anderes…

2005: MSBuild wird aus VisualStudio „herausgelöst“, seit 2015 quelloffen

- Standard für .NET Entwicklung

2012: gradle in der Welt von Java und anderen

- Benutzt Groovy-basierte DSL (anstatt XML)
- Standard bei google, zB AndroidStudio

Beispiele zum Ausprobieren:
https://technologyconversations.com/2014/06/18/build-tools/

Und was brauchen wir?

- Ziele (Anwendungsfälle in Aufg. 4)
  - Verschiedene Ein- und Ausgabe
  - Unittest und Integrationstest

	Eingabe	Ausgabe
1	GUI	GUI
2	HiL	HiL
3	HiL	GUI
4	HiL	NXT
5	HiL	EV3
6	EV3	EV3
7	NXT	NXT

- Abhängigkeiten: java Dateien, Bibliotheken, angeschlossene Hardware
- Befehle: Java-Compiler, nxj-Befehle und Start verschiedener Main-Klassen

Und was brauchen wir jetzt wirklich?

Rahmenbedingungen:

- Ein Entwickler
- Lokaler Rechner
- Keine komplexen Abhängigkeiten

→Ausführung von

a) Test (Unit- und Integrationstests)

b) Applikation mit unterschiedlicher In- und Out-Schnittstelle

Mein Tipp (v.a. für Anfänger): Ant (ivy ist nicht erforderlich)
Wer mehr lernen möchte: Gradle
(Maven ist eher ungeeignet wegen NXT)